What is a Microprocessor?

- The word comes from the combination Micro (indicating very small size) and Processor (process means to manipulate e. g. certain operations on the numbers).
- Definition of the Microprocessor

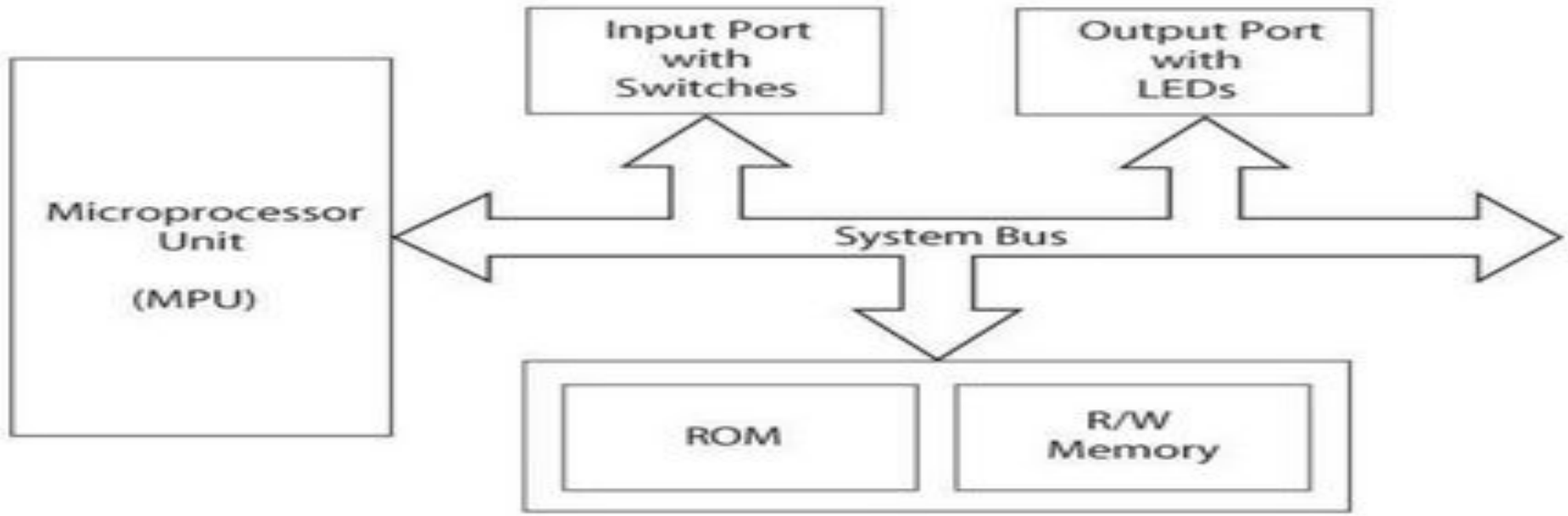
  A programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result

Difference between:

- Microcomputer – a computer with a microprocessor as its CPU. Includes memory, I/O etc.
- Microprocessor – A silicon chip which includes ALU, registers (as a small internal memory) & a control unit.
  - ✓ A general purpose device (i.e. may be used for different purposes in different applications e.g. as a CPU in a microcomputer)
  - ✓ Uses Memory, I/O functions etc external to the chip
  - ✓ Configuration of the system is flexible
- Microcontroller - A silicon chip which includes microprocessor, memory & I/O etc & other peripheral functions all integrated on a single chip.
  - ✓ More application specific, single purpose
  - ✓ Especially used in embedded systems
  - ✓ Fixed configuration (i.e. fixed memory & other functions)

A Microprocessor-based system

A microprocessor based system (e.g. a microcomputer) consists of the following components:



Inside a Microprocessor

- Internally, the microprocessor is made up of 3 main units.
  - The Arithmetic/Logic Unit (ALU)
  - The Control Unit.
  - An array of Registers as a small internal memory for holding data while it is being manipulated or processed.

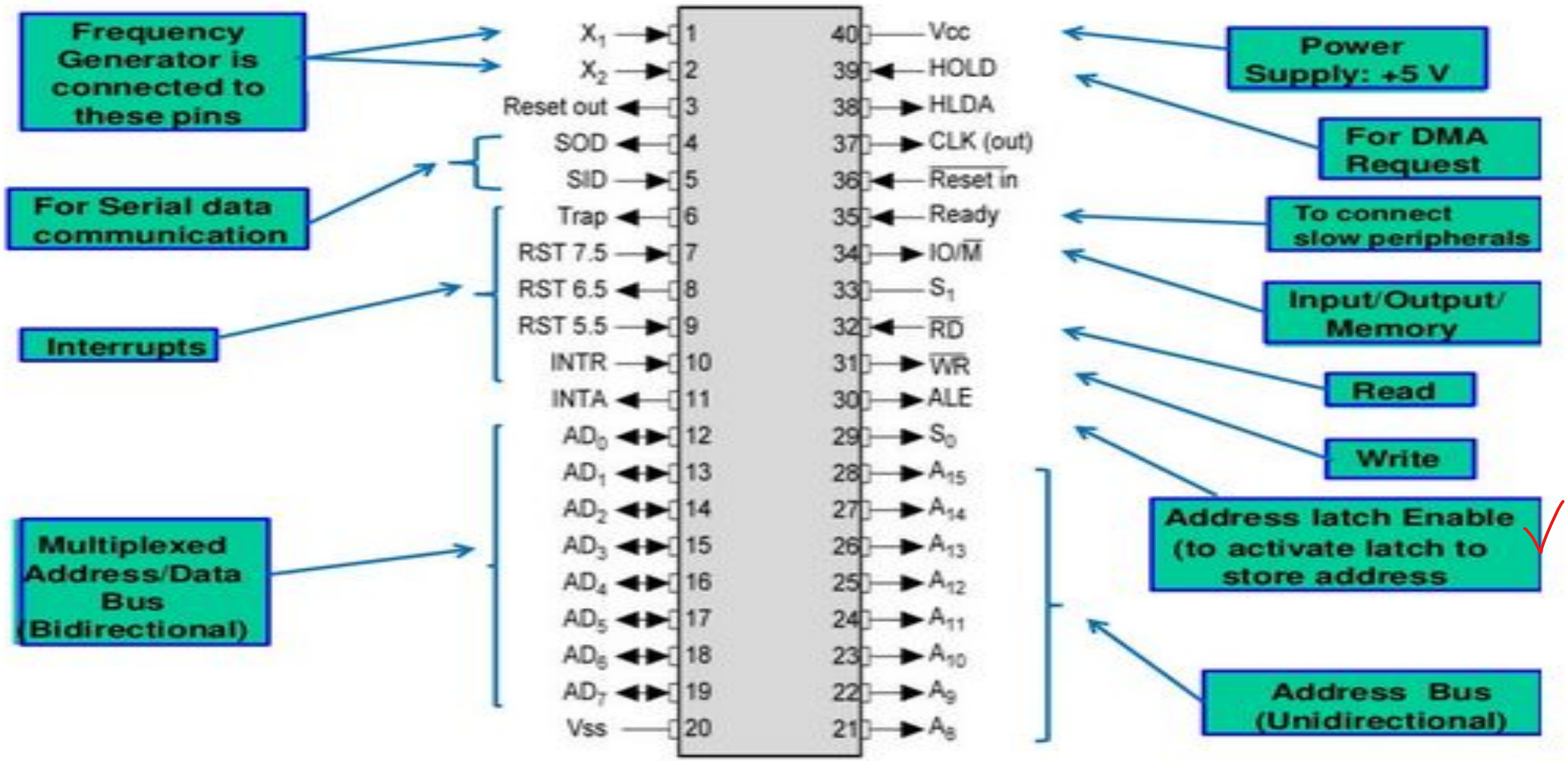
Memory of Microprocessor

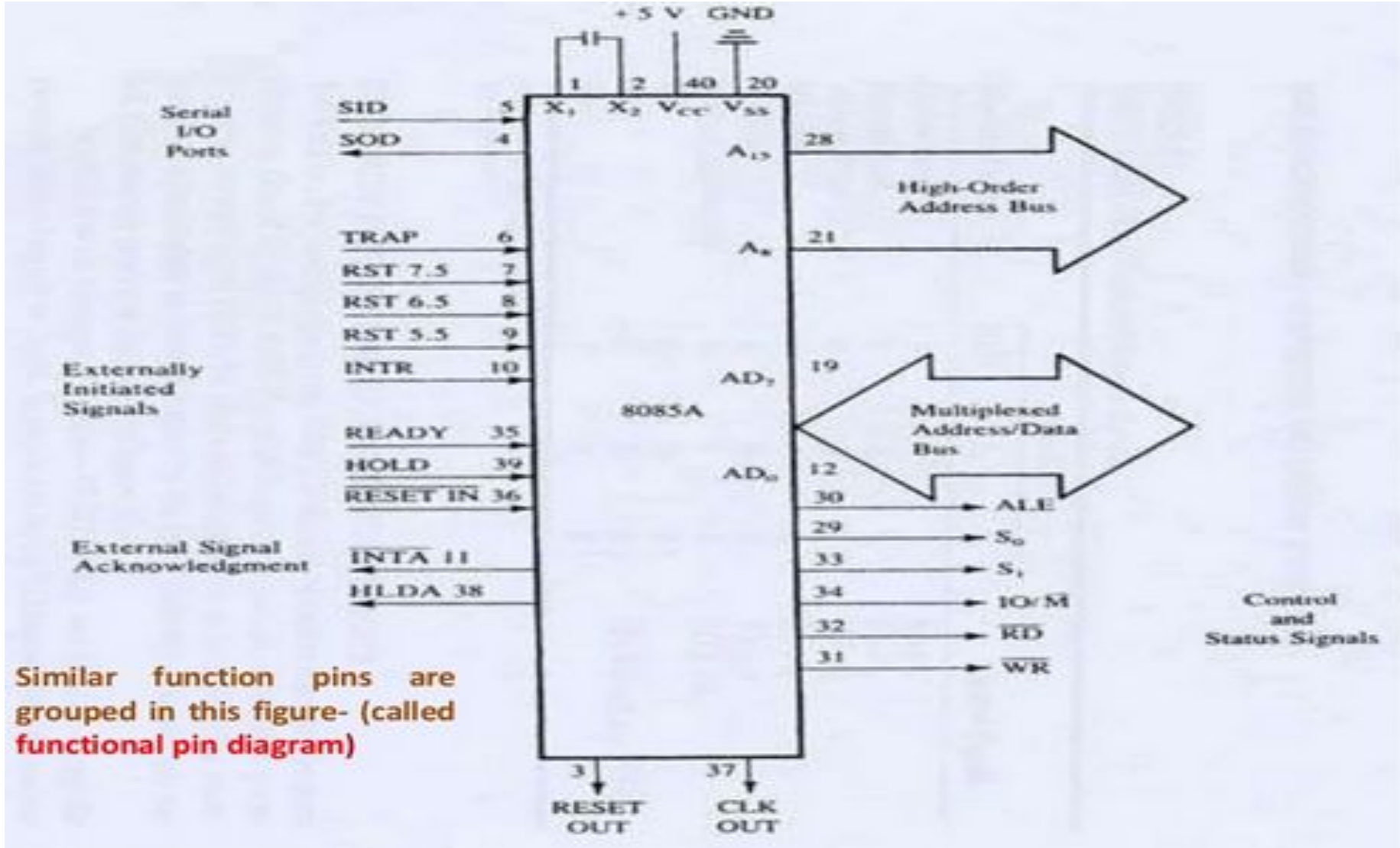
- **Inside (very small)**
  - The registers inside the microprocessor

- **Outside**
  - Read Only Memory (ROM)
    - used to store information that does not change
  - Random Access Memory (RAM)
    - used to store information supplied by the user such as programs and data.

8085 Microprocessor

- One of the most popular 8-bit general purpose µp launched by Intel, USA in 1976
- An N-MOS chip with 40 pins & +5V supply
- Capable of addressing $2^{16} = 64$ KB of memory
- It works on 3 MHz clock.
- It has multiplexed address and data bus (AD0-AD7) to reduce hardware (no. of pins) on the chip
- It has 74 basic instructions (formats) with 5 different addressing modes.


Frequency Generator is connected to these pins
For Serial data communication
Interrupts
Multiplexed Address/Data Bus (Bidirectional)
Power Supply: +5 V
For DMA Request
To connect slow peripherals
Input/Output/ Memory
Read
Write
Address latch Enable (to activate latch to store address
Address Bus (Unidirectional)
X1 1
X2 2
Reset out 3
SOD 4
SID 5
Trap 6
RST 7.5 7
RST 6.5 8
RST 5.5 9
INTR 10
INTA 11
AD0 12
AD1 13
AD2 14
AD3 15
AD4 16
AD5 17
AD6 18
AD7 19
Vss 20
40 Vcc
39 HOLD
38 HLDA
37 CLK (out)
36 Reset in
35 Ready
34 IO/M
33 S1
32 RD
31 WR
30 ALE
29 S0
28 A15
27 A14
26 A13
25 A12
24 A11
23 A10
22 A9
21 A8



Similar function pins are grouped in this figure- (called functional pin diagram)

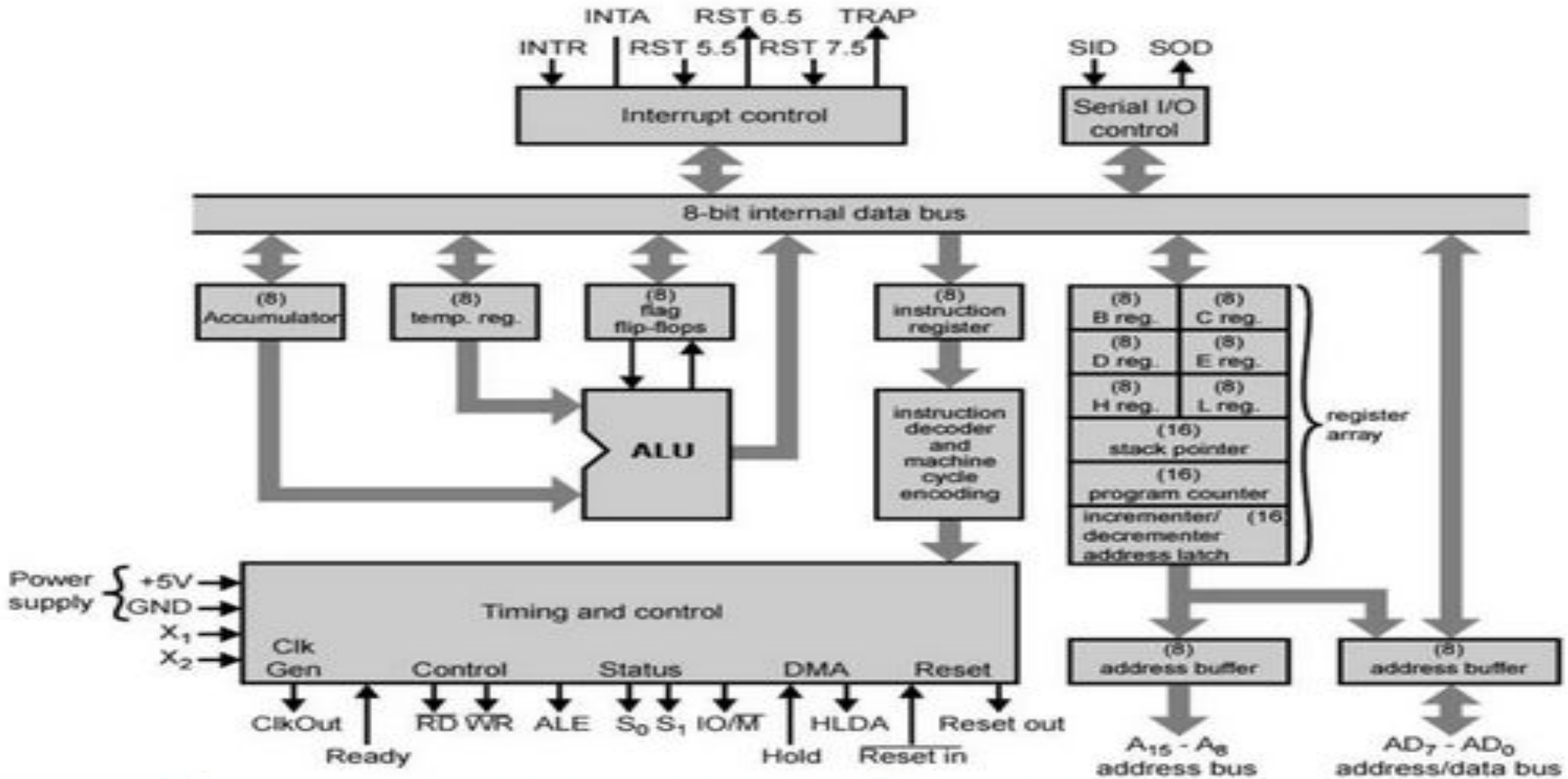
PIN FUNCTIONS

- **AD0-AD7:** Lower order bidirectional multiplexed Address/Data lines.
- **A8-A15:** Higher order unidirectional address lines, carries only address
- **ALE:** A pulse output signal from this pin is used by microprocessor to enable an 8-bit external latch to save the lower order address bits
- **S0,S1:** These output status signals are used to indicate type of operation
- **RD:** MP reads data from memory/IO device when this pin is active (low)
- **WR:** MP writes data into memory/IO device when this pin is active (low)
- **READY:** This pin is used by slow-responding peripheral devices to indicate the µP whether they are READY to send/accept data to/from µP
- **TRAP:** A highest priority , non mask able vectored interrupt. After TRAP, restart occurs and execution starts from predefined vector address 0024H
- **RST5.5,6.5,7.5:**These are maskable, vectored interrupts and have lower priority than TRAP
- **INTR:** INTR is a lowest priority non-vectored which can be used to connect upto 8 peripheral devices by using an interrupt controller
- **INTA:** An interrupt acknowledge signal pin for INTR, only acknowledgment pin among all interrupts

PIN FUNCTIONS

- **HOLD & HLDA:** When a peripheral device wants to gain control of system buses (e.g. for a DMA operation), it sends request to µP via this input pin. In response, µP activates HLDA (output) to acknowledge the request & temporarily releases control over system buses.
- **IO/M:** This output control pin is used to indicate whether the read/write operation being performed by microprocessor is for memory (when low) or for I/O device (when high)
- **RESET IN:** This control input signal connected to external RESET button to bring µP in initial standby mode. It restarts µp to memory location 0000H.
- **RESET OUT:** When high, this indicates µP has been reset
- **SID & SOD:** These pins are used for serial data communication by using making use of SIM & RIM instructions
- **X1X2** :These are clock input signals which are connected to external oscillator of 6 MHz frequency which is divided by 2 internally in 8085 to generate 3 MHz operating frequency .
- **CLK (out):** Used to provide same 3MHz clock to the rest of the system
- **VCC, VSS & GND:** Power supply pins VCC= +5V & VSS=GND=0V

Architecture of 8085 Microprocessor



Functions of various building blocks:

ALU: Arithmetic & Logic Unit contains digital circuitry to perform all arithmetic & logical operations in 8085

TIMING & CONTROL UNIT: Generates various types of control signals to direct microprocessor what & when a task is to be performed.

INSTRUCTION DECODING & MACHINE CYCLE ENCODING: Used to decode instruction opcode into binary form & to control various machine cycle operations during execution of an instruction

SERIAL I/O CONTROL: Used to control serial data communication with external devices via SID/SOD lines

INTERRUPT CONTROL: Used to handle various incoming interrupts which occur via 5 interrupt pins TRAP, RST7.5, RST6.5, RST5.5 & INTR lines

INCREMENTER/DECREMENTER ADDRESS LATCH: It increments/ decrements contents of Program Counter or Stack Pointer when instructions related to them are executed.

ADDRESS & DATA BUFFERS: These are used to momentarily hold the 16 'in-transit' address bits while they are being placed on Address/Data bus. These effectively act as an 'interface' between internal & external buses.

GENERAL PURPOSE REGISTERS (B,C,D,E,H,L): These 6 register can be used singly to store any 8-bit data or as register pairs (valid pairs are B-C, D-E & H-L) to perform 16-bits operations.

TEMPORAY REGISTERS (W, Z & TR): Used internally by 8085 to temporarily store 8-bit data (or partial results) while performing various arithmetic operations. Not available to user

ACCUMULATOR: An 8 bit register having some special features (usually included in ALU block). It stores results of arithmetic & logical operations. Many instructions in 8085 are Accumulator-based.

INSTRUCTION REGISTER: An 8-bit register used to hold opcode of the instruction currently being executing

PROGRAM COUNTER: Holds 16-bits address of the next byte to fetched during execution of an instruction & is incremented by 1 each time a byte is fetched to point to the next memory location.

STACK POINTER: A 16-bits register which holds the address of topmost memory location of stack. It decrements/increments by 2 each time a PUSH/POP operation is performed

FLAG REGISTER: An 8 bit register whose individual bit are set/reset according to status of result generated by ALU.

FLAG REGISTER

S	Z	X	AC	X	P	X	CY

Set = 1, Reset = 0

S: Sign flag is set when result of an operation is negative (leftmost bit is 1 in signed arithmetic operations)

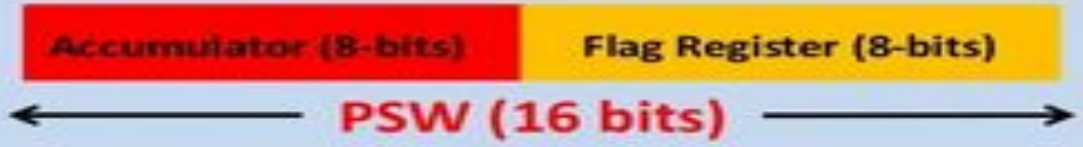
Z: Zero flag is set when result of an operation is zero otherwise reset

AC: Auxiliary carry flag is set when there is a carry out of 3rd to 4th bit in an 8-bit operation

CY: Carry flag is set when there is a final carry generated in an operation.

P: Parity flag is set when result contains even number of 1's (Reset for odd number of 1's)

Program Status Word: The combined 16-bits contents of Accumulator & Flag register (A & F) are named as PSW



INSTRUCTION FETCH & EXECUTION OPERATION IN 8085

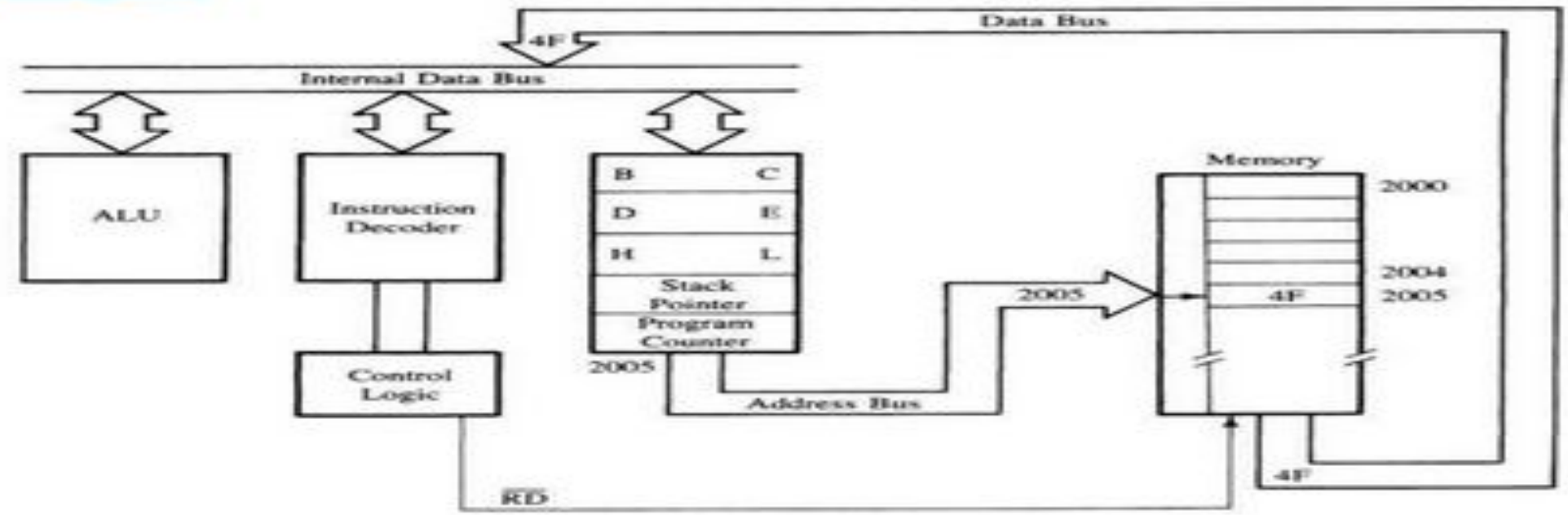
Procedural steps of an example instruction (MOV C, A) (Hexcode: 4F) being fetched & executed by microprocessor from memory location 2005:

Step 1: 8085 places 16-bits address (2005H) from program counter on address bus

Step 2: Control unit activates $\overline{RD}$ signal to enable memory chip (RAM)

Step 3: Instruction byte (4F) is placed by memory on data bus

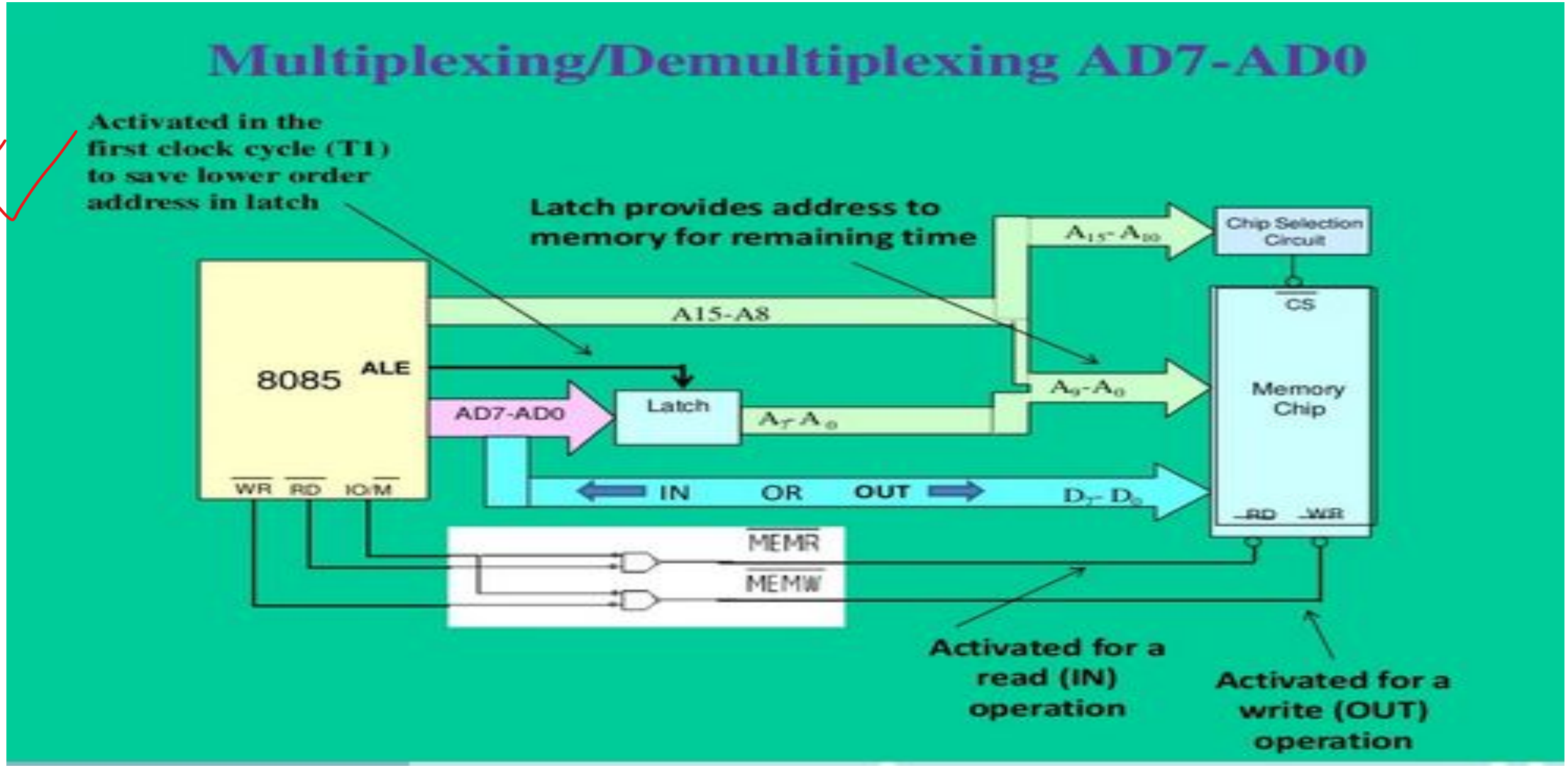
Step 4: Instruction is decoded by Instruction decoder in binary form which directs timing & control unit to generate appropriate control signals to carry out the task as specified by the instruction



Multiplexing/Demultiplexing AD7-AD0

- AD7-AD0 lines serve dual purpose; they work in time-shared mode: at one time they carry address & data at other time, to save 8 extra pins on the chip
- For a read/write operation from memory, we must provide complete 16-bit address for full duration (3 clock periods) of read/write cycle (until data is loaded to/from memory.
- The 8 higher-order bits of the address remain on the bus for full read/write cycle, but low-order bits remain for only 1 clock period (to make data bus free to carry data later). So we use an external 8-bit latch (a chip) to save & hold lower-order address for the remaining time (2 clock periods) or else it (lower order address) would be lost.
- We use ALE control signal to activate (to save lower order address) latch in the first clock period.

Multiplexing/Demultiplexing AD7-AD0



The 8085 Instructions (Summary)

The 8085 instructions can be categorized into 5 different groups:

- Data Transfer
- Arithmetic
- Logic
- Branch
- Machine Control

INSTRUCTION WORD SIZE IN 8085

- **One-Byte Instructions:** Occupies 1 byte in the memory, no 8/16 bits data is included in the instructions

  Examples: MOV A, B ADD B RAL MOV C, M
  CMC STC CMP M ANA B INR C

- **Two-Bytes Instructions:** Occupies total 2 bytes in the memory, 8-bits data is included in the instructions

  Examples: MVI A, 05 IN 02 CPI 03

- **Three-Bytes Instructions:** Occupies total 3 bytes in the memory, 16-bits data is included in the instructions

  Examples: LXI H, 2500 LDA 3050 LHLD 2600

8085 ADDRESSING MODES

Addressing Mode: An addressing mode indicates the location of operand in an instruction

1. **IMMEDIATE:** Operand itself is immediately given in the instruction
   Examples: MVI A,05H LXI H, 2050H ADI 02
2. **DIRECT:** 16 bits address (or 8 bit in case of I/O instructions) is directly given in the instruction itself
   Examples: LDA 2500H IN 06 SHLD 2605 OUT 03
3. **REGISTER:** Operand is given in one of the registers
   Examples: MOV B,C DCR C SUB D ANA H ADD B
4. **REGISTER INDIRECT:** 16-bits address of operand is indirectly provided in one of the register pairs
   Examples: MOV A, M ADD M INR M LDAX B STAX D
5. **IMPLICIT (or IMPLIED):** Operand is in the Accumulator
   Examples: RAL CMA RRC RLC

MACHINE CYCLES & TIMING DIAGRAMS

Instruction Cycle: Total time required to execute an instruction. Includes opcode fetch & execution cycles

Machine Cycle: Time required by µP to complete one operation (read/write/fetch) of accessing memory or I/O devices during execution of an instruction. Machine cycles are of fetch, memory read/write & I/O read/write type. First machine cycle is always an opcode fetch cycle in an instruction cycle. An instruction in 8085 may have minimum 1 (which is a fetch cycle) to a maximum of 5 machine cycles depending on type of instruction.

T-State: A subdivision of an operation (read/write/fetch) carried out in one system clock period. In 8085, while a fetch cycle in all instructions requires 4 T-states, other machine cycles may require minimum 3 to a maximum of 6 T-states depending upon type of instruction.

Timing Diagram: A timing diagram is a graphical representation of various control signals generated during execution of an instruction. It depicts status of important control signals in each T-state of all machine cycles involved in the execution of an instruction.

THE STACK

What is Stack?

- The stack is a group of memory locations set aside by user in memory area for temporary storage (i.e. it is a portion of RAM itself) because 8085 has limited no. of registers. In addition, it has some special features.
- Starting address of stack (initialization of stack) is defined by user by loading a 16-bit address into Stack Pointer as below:

LXI SP, XXXX

where XXXX is any 16-bits address chosen by user

When is Stack used?

1. Stack is used to save the return address of Program Counter when:

- ✓ *A CALL/RSTn instruction (a software interrupt) encounters in a program*
- ✓ *A hardware interrupt occurs during execution of a program*

In any of the above cases:

i) µP executes the current instruction at hand & saves address of the next instruction (i.e. contents of PC, the return address) in stack so that it knows where it was before jumping to the subroutine/ISR of the respective interrupt.

ii) Microprocessor loads PC with starting address of subroutine (in case of a CALL instruction) or ISR (in case of a hardware interrupt)
iii) Microprocessor executes respective subroutine/ISR
iv) Microprocessor retrieves back the saved return address from the stack (by using RET instruction, which is the last instruction of all subroutines) and loads back it into PC.
v) Microprocessor re-continues the main program as usual

➢ In the above cases, use of stack is made by µP internally. User only defines its starting address

2. Stack can also be used by user 'manually' to temporarily store the contents of a register pair (& not a single register) so that it be used for other operations or purposes. For example, consider HL pair has some specific address or data stored in it at the start of a program. Now a user wants to 'free' the HL pair for using in some other instructions in the program but at the same time doesn't want to lose the initial contents. For this, he can save contents of HL pair in stack, use HL pair in other instructions & then get back saved contents from stack. Storage (called PUSHing) & retrieval (called POPing) of any contents to/from stack is achieved by using PUSH & POP instructions respectively.

STACK PRINCIPLES

- The storage and retrieval of any content on the stack follows the Last-In-First-Out (LIFO) sequence.
- The stack normally grows backwards into memory
- Stack Pointer is a 16-bits register located in 8085 chip, which holds the address of topmost location (called Stacktop) of stack. The contents of SP are decremented/incremented by 1 with storage/retrieval of a each byte into stack
- Although stack can be initialized anywhere in the memory area, usually it is initialized at the highest available memory location so that chances of interference between stack locations & user program are minimized because address in Stack Pointer is decremented with each byte storage into stack i.e. stack will grow backwards into memory (towards user program)
- The first location available for a write (or PUSH) operation in stack is one that is addressed by SP-1 & not the SP. However, the first location for a read (or POP) operation is one that is addressed by SP. During PUSHing, the stack operates in a "Decrement then Store" style i.e. decrement SP first then store at the new (lower) location address as indicated by SP. During POPing, the stack operates in a "Use then Increment" style i.e. read the contents of topmost location (addressed by SP) first & then increment it to indicate towards a new (higher) topmost location.

Execution of PUSH H

Execution of PUSH H instruction, as explained in the previous example, is illustrated here:

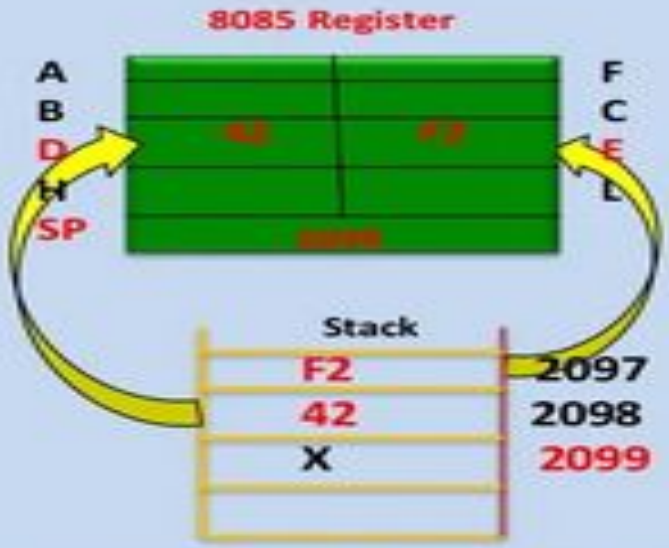
- The SP is decremented by one to 2098H, & the contents of the H register (42H) are copied to memory location 2098H
- The SP is again decremented by one to 2097H & the contents of the L register (F2H) are copied to memory location 2097H
- New SP contents after PUSH operation are 2097

Contents of SP & stack locations after PUSH H operation

Execution of POP D

Execution of POP D instruction, as explained in the previous example, is illustrated here:

- The contents (F2H) of the current stacktop location (2097H) are copied to E register and the SP is incremented by one to 2098H
- The contents of the new top location (2098H) of the stack are copied to D register, and the SP is incremented by one again
- Address of new topmost location (contents of SP) now becomes 2099H after the POP operation



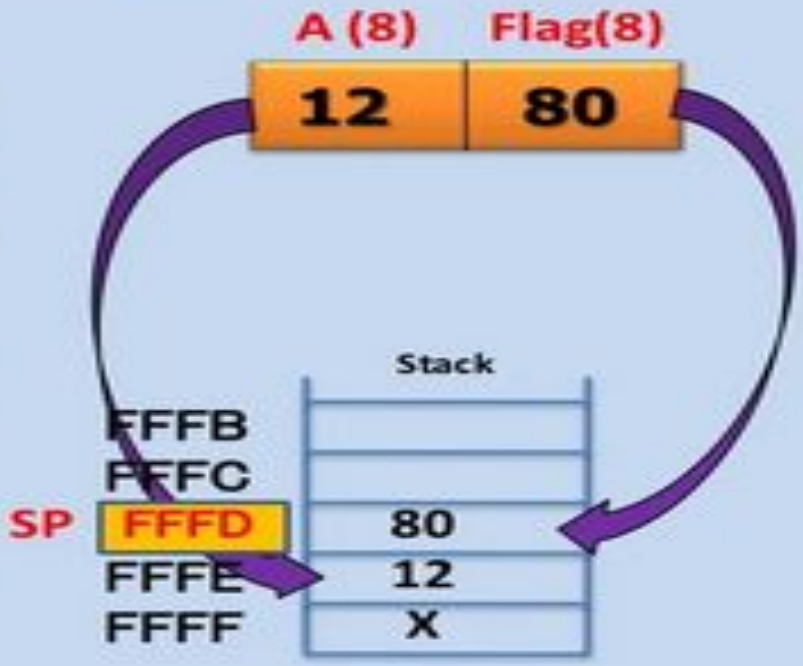
Contents of SP & stack locations after POP D operation

PUSH PSW

PUSH PSW instruction is similar to **PUSH Rp** instruction except that it stores PSW (the 16-bits combined contents of Accumulator & Flag Register) instead of a general purpose register pair into the stack

For example assume Accumulator & Flag registers contain 12H & 80H respectively. Assume stack is initialized at FFFFH (i.e. contents of SP) then on execution of PUSH PSW:

- SP is decremented by one & the contents of Accumulator (12H) are copied first to the FFFEH location
- SP is decremented by one again & the contents of Flag register (80H) are copied then to the FFFDH location
- New contents of SP are FFFDH after the PUSH PSW operation.



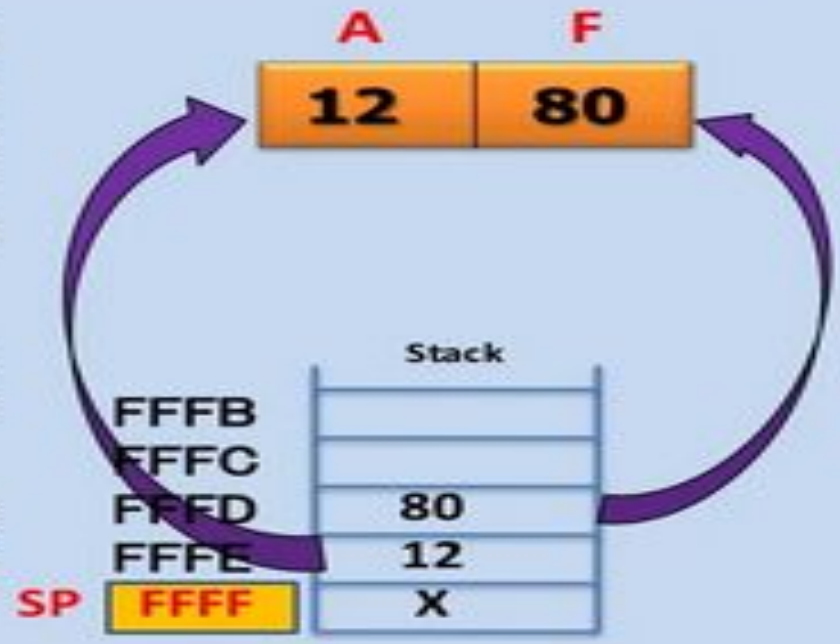
Contents of SP & PSW after PUSH PSW operation

POP PSW

POP PSW instruction is similar to POP Rp instruction except that it reads contents of top two locations from the stack & loads them into PSW (or Accumulator & Flag Register)

Continuing with example of PUSH PSW explained earlier, we assume that current address in SP is FFFDH having value 80H. Then, on execution of POP PSW following actions take place in steps:

- Contents of top location addressed by SP (i.e. 80H) are copied to the Flag register first.
- SP is incremented by one & the contents of current top location (now FFFEH) are copied to the Accumulator.
- SP is incremented by one again & the contents of current top location now become FFFFH.



Contents of SP & PSW after POP PSW operation

SUBROUTINE

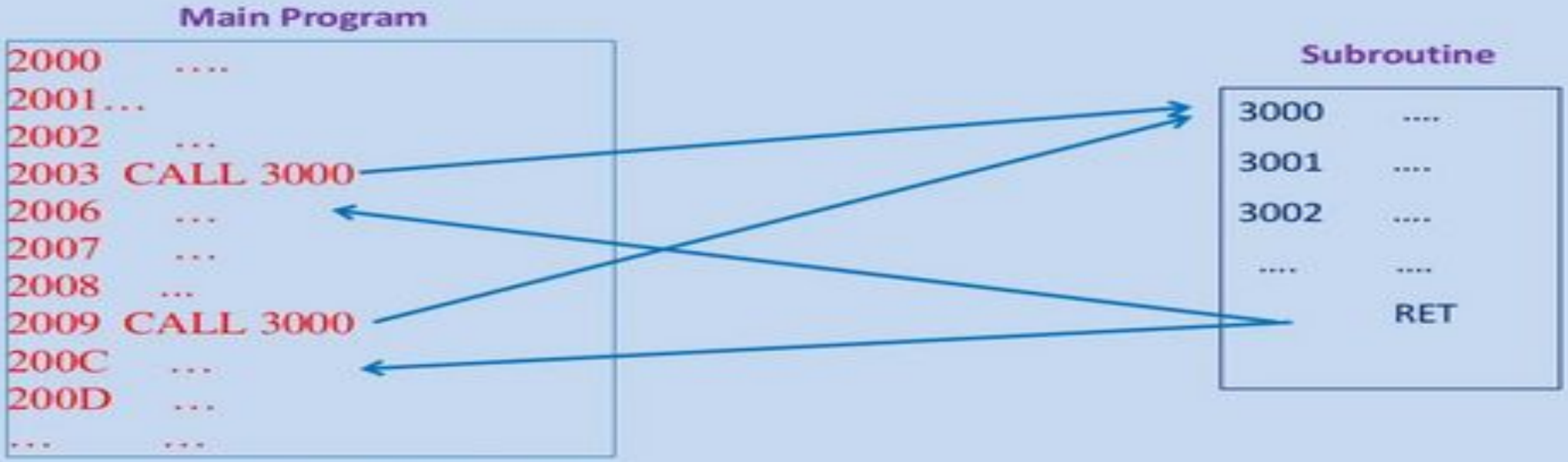
- A subroutine is group of instructions (a subprogram) written separately from the main program to perform a function that occurs repeatedly in the main program
- When a main program CALLs a subroutine, the program execution is transferred to the subroutine. After the completion of the subroutine ,the program execution returns to the main program.
- The microprocessor uses stack to store the return address.
- 8085 has two instructions for dealing with subroutines:
  - The CALL instruction, used to redirect program execution to the subroutine. Format of CALL instruction is:
    CALL XXXX (where XXXX is 16-bits starting address of the subroutine)
    A CALL instruction may be of conditional or unconditional type
  - The RET instruction, used to return to the main program at the end of the subroutine. No address is provided by user in a return instruction as it automatically retrieves return address from the stack top locations. RET may also be of conditional or unconditional type
- A CALL instruction is always used in conjunction with RET instruction

EXECUTION OF CALL/RET INSTRUCTIONS

Following sequence of actions take place in when microprocessor encounters a CALL instruction in the middle of a program (same steps are taken by microprocessor when a hardware interrupt occurs in the middle of program):

- ✓ Microprocessor saves the address of next instruction (i.e. contents of PC, the return address) in the stack so that it knows where it was before jumping to the subroutine.
- ✓ Microprocessor loads PC with 16-bits starting address (provided in the instruction) of the subroutine
- ✓ Microprocessor executes the subroutine
- ✓ Microprocessor retrieves back the saved return address from the stack (by using RET instruction, the last instruction of subroutine) and loads back it into the PC
- ✓Microprocessor re-continues the main program as usual

Consider the following situation in which microprocessor is executing a (main) program (stored at 2000H onwards in RAM) & faces CALL instructions at 2003H & 2009H locations. Assuming subroutine is stored 3000H onwards, microprocessor saves address of next instructions (2006H & 200CH respectively) in the stack, loads starting address (3000H) of subroutine into PC & executes the subroutine. At the end of the subroutine, RET instruction instructs microprocessor to retrieve the saved return addresses which are then loaded back into PC & thus microprocessor continues the main program as before.



8085 INTERRUPTS

- An interrupt is a hardware or software initiated subroutine CALL.
- An interrupt is considered to be an emergency signal that is serviced by the Microprocessor as soon as possible.

- What happens when MP is interrupted ?

  When microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to a new memory location (interrupt vector) to execute an Interrupt Service Routine (ISR) in response to the incoming interrupt. There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.

  Vectored: The address of the subroutine is already known to the microprocessor

  Non Vectored: The external device supplies the address of the subroutine to the Microprocessor

➢ **Interrupts are also categorized as SOFTWARE & HARDWARE interrupts. In 8085:**

SOFTWARE Interrupts: RST n & CALL instructions

HARDWARE Interrupts: TRAP, RST 7.5, 6.5, 5.5, INTR

- An interrupt vector is a pointer or memory address where the ISR is stored in the memory
- All interrupts are mapped onto a particular memory area called the Interrupt Vector Table (IVT) which shows vector addresses for each interrupt
- For each type of interrupt, a unique ISR is written at respective vector locations which directs microprocessor to take some predefined action (s)
- An RSTn instruction is equivalent to a CALL instruction but with a fixed address & occupies only a 1byte as compared to 3bytes by a CALL instruction

Pin or interrupt	Predefined Vector addresses for 8085 hardware interrupts
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C
INTR	*

** the address of the ISR is determined by the external hardware (or interrupt controller)*

RSTn Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

Execution of an Interrupt

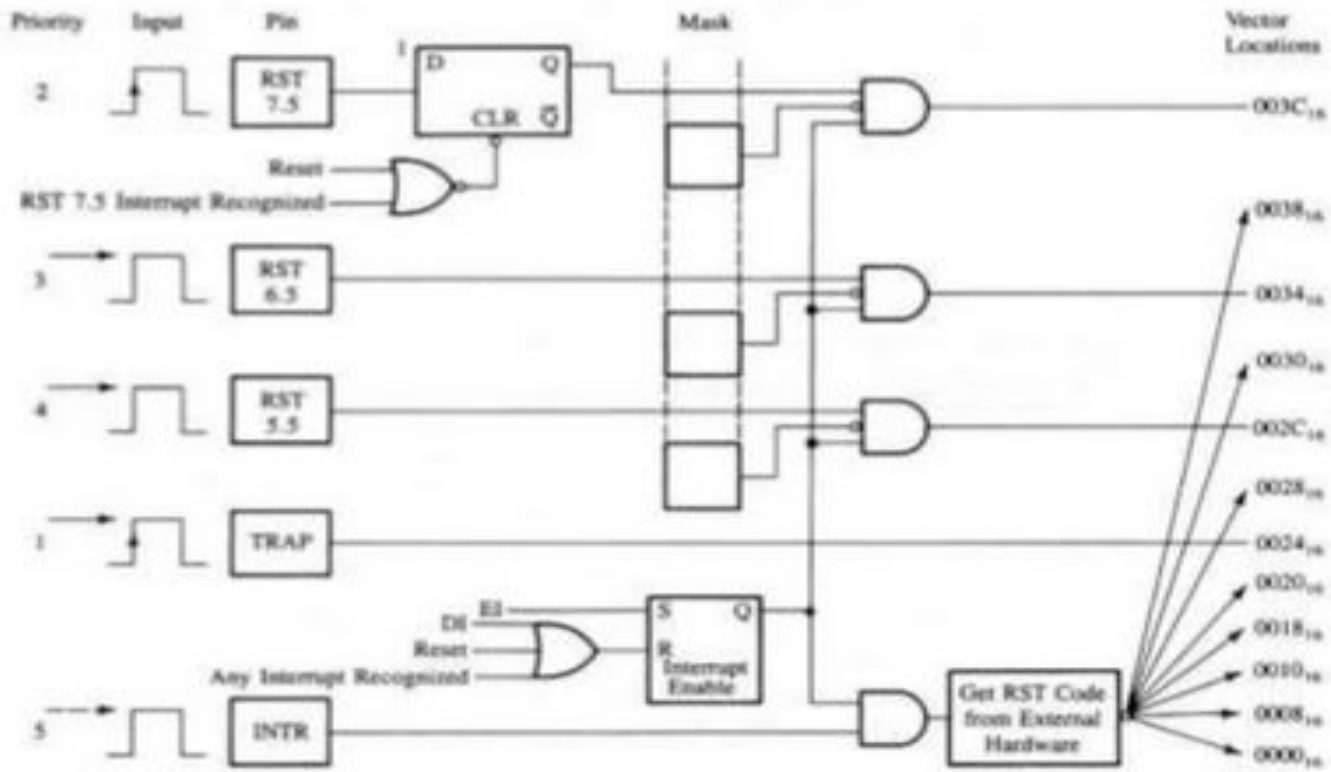
Upon receiving an Interrupt signal, microprocessor saves the memory location of the next instruction on the stack and the program is transferred to a new memory location specified by its vector address where microprocessor executes an ISR. All other maskable interrupts are automatically disabled until current interrupt is attended

- An ISR is a predefined set of instructions (a subprogram) which performs a specific task. It must include the EI instruction in one of the ending instructions so that all interrupts are re-enabled for coming in future
- Last instruction of an ISR must always be an RET instruction which instructs microprocessor to retrieve the saved return address from the stack so that program is transferred back to where the program was interrupted.

COMPARISON OF 8085 HARDWARE INTERRUPTS

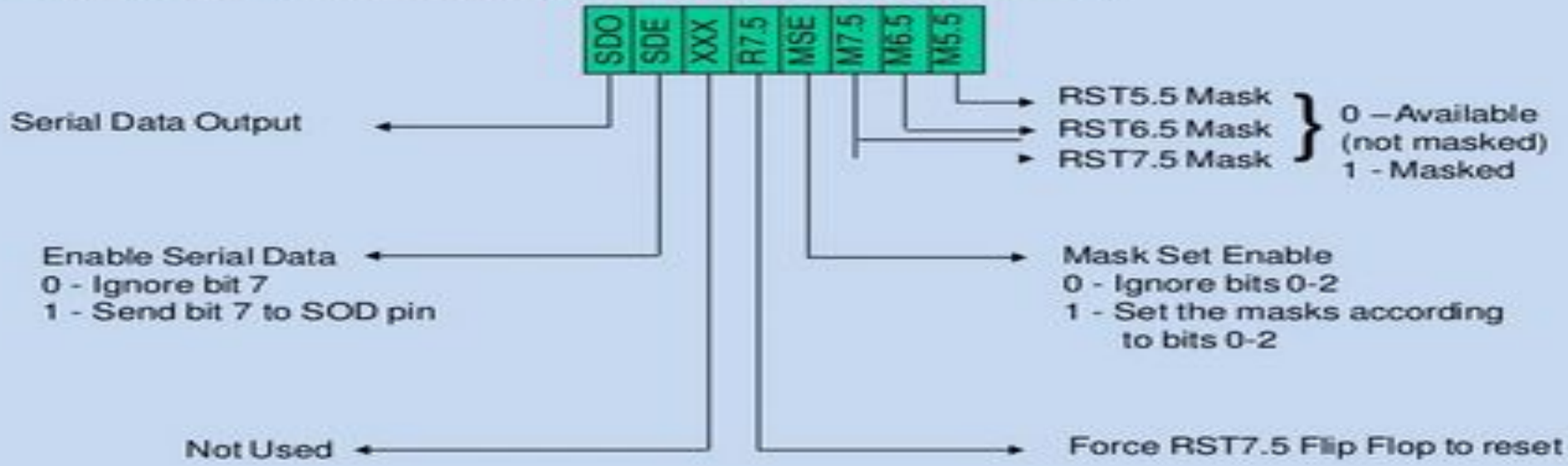
Interrupt Name	Priority	Maskable	Masking Method	Vectored	Triggering Method
INTR	Lowest	Yes	DI / EI	No	Level Sensitive
RST 5.5 / RST 6.5		Yes	DI / EI SIM	Yes	Level Sensitive
RST 7.5		Yes	DI / EI SIM	Yes	Edge Sensitive
TRAP	Highest	No	None	Yes	Level & Edge Sensitive

HARDWARE INTERRUPT BOCK DIAGRAM



SIM Instruction

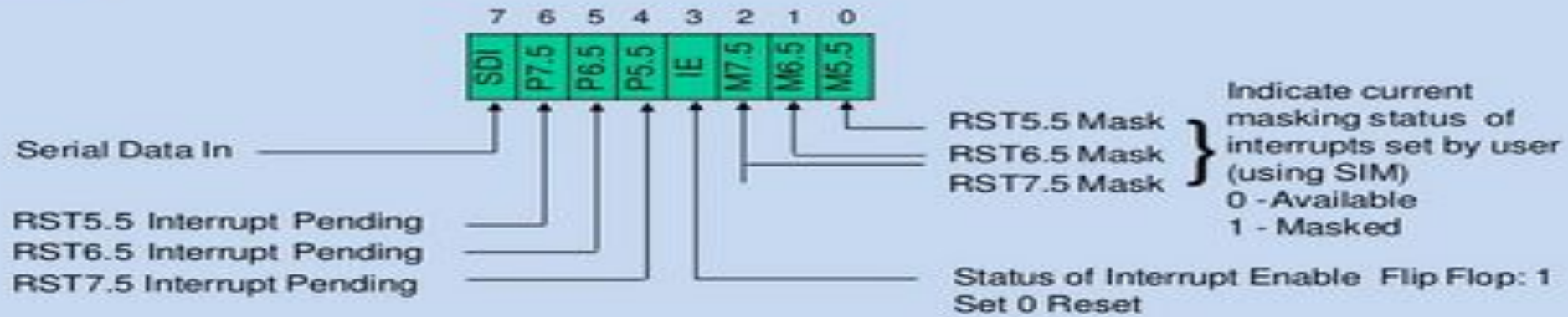
SIM instruction can be used to perform two different tasks: 1. For masking of 3 interrupts 2. For serial data transmission (Each time a SIM instruction is executed, 7th bit of Accumulator is automatically copied to SOD pin of 8085)



While EI/DI instructions enable/disable all maskable interrupts at once, SIM instruction can be used to selectively mask (or disable) 3 out of 4 maskable interrupts which are RST7.5,RST6.5 & RST5.5. Fourth maskable interrupt INTR can only be enabled/disabled by using EI/DI instructions.

RIM instruction

Like SIM instruction, RIM can be used to perform two different tasks: 1. To read current status of 3 maskable interrupts 2. For serial data reception (Each time a SIM instruction is executed, the bit present on SID pin of 8085 is automatically moved to 7th bit of the Accumulator)



Pending Interrupts: Since the 8085 has 5 interrupt lines, another interrupts may occur while an interrupt is being attended and thus remain pending. Such interrupts are called pending interrupts & would be attended as soon as ISR of current interrupt is executed. A programmer may know the status (current value of high/low on the respective interrupt pin) of such interrupts anytime by using RIM instruction.

Features	Memory Mapped IO	IO Mapped IO
Addressing	IO devices are accessed like any other memory location.	They cannot be accessed like any other memory location.
Address Size	They are assigned with 16-bit address values.	They are assigned with 8-bit address values.
Instructions Used	The instruction used are LDA and STA, etc.	The instruction used are IN and OUT.
Cycles	Cycles involved during operation are Memory Read, Memory Write.	Cycles involved during operation are IO read and IO writes in the case of IO Mapped IO.
Registers Communicating	Any register can communicate with the IO device in case of Memory Mapped IO.	Only Accumulator can communicate with IO devices in case of IO Mapped IO.

Features	Memory Mapped IO	IO Mapped IO
Space Involved	2^{16} IO ports are possible to be used for interfacing in case of Memory Mapped IO.	Only 256 I/O ports are available for interfacing in case of IO Mapped IO.
IO/M` signal	During writing or read cycles (IO/M` = 0) in case of Memory Mapped IO.	During writing or read cycles (IO/M` = 1) in case of IO Mapped IO.
Control Signal	No separate control signal required since we have unified memory space in the case of Memory Mapped IO.	Special control signals are used in the case of IO Mapped IO.
Arithmetic and Logical operations	Arithmetic and logical operations are performed directly on the data in the case of Memory Mapped IO.	Arithmetic and logical operations cannot be performed directly on the data in the case of IO Mapped IO.

Basis for Comparison	Memory mapped I/O	I/O mapped I/O
Basic	I/O devices are treated as memory.	I/O devices are treated as I/O devices.
Allotted address size	16-bit (A_0 – A_{15})	8-bit (A_0 – A_7)
Data transfer instructions	Same for memory and I/O devices.	Different for memory and I/O devices.
Cycles involved	Memory read and memory write	I/O read and I/O write
Interfacing of I/O ports	Large (around 64K)	Comparatively small (around 256)
Control signal	No separate control signal is needed for I/O devices.	Special control signals are used for I/O devices.
Efficiency	Less	Comparatively more
Decoder hardware	More decoder hardware required.	Less decoder hardware required.

Basis for Comparison	Memory mapped I/O	I/O mapped I/O
IO/M’	During memory read or memory write operations, IO/M’ is kept low.	During I/O read and I/O write operation, IO/M’ is kept high.
Data movement	Between registers and ports.	Between accumulator and ports.
Logical approach	Simple	Complex
Usability	In small systems where memory requirement is less.	In systems that need large memory space.
Speed of operation	Slow	Comparatively fast
Example of instruction	LDA ****H STA ****H MOV A, M	IN ****H OUT ****H